Assignment 2

Edvin Magnusson

Question 1

Logical Equality

```
logical_equality <- function(A,B){

  A<- as.logical(A)
  B<- as.logical(B)

  AB<- A==B

  return(AB)
}

logical_equality(A = TRUE, B = FALSE)
```

```
## [1] FALSE
```

```
logical_equality(A = FALSE, B = FALSE)
```

```
## [1] TRUE
```

```
logical_equality(A = T, B = T)
```

```
## [1] TRUE
```

Question 2

Sheldon game (Rock-paper-scissors-lizard-spock).

```
sheldon_game <- function(player1, player2) {
  choices <-
    c("Rock", "Paper", "Scissors", "Lizard", "Spock") # The valid schoices
  stopifnot(player1 %in% choices, player2 %in% choices) # Stop if not a valid choice
  choice1 <- which(choices %in% player1) # choice for player 1
  choice2 <- which(choices %in% player2) # choice for player 2
  if (any((choice1 == 1 &
           choice2 ==  (3 |
                          4)) |
          (choice1 == 2 &
           choice2 == (1 |
                         5)) | (choice1 == 3 & choice2 == (2 | 4)) | # The Sheldon rules
          (choice1 == 4 &
           choice2 == (5 | 2)) | (choice1 == 5 & choice2 == (1 | 3))
  )) {
    return("Player 1 wins!")
  } else if (choice1 == choice2) {
    return("Draw!")
  } else {
    return("Player 2 wins!")
  }
}

sheldon_game("Paper", "Rock")
```

```
## [1] "Player 1 wins!"
```

```
sheldon_game("Spock", "Spock")
```

```
## [1] "Draw!"
```

Question 3

Moving Median

```
#Using the ellipse ... to have unspecified variables

my_moving_median <- function(x,n,...){

  if(!is.numeric(x)|!is.numeric(n)){
    stop("x or n is not numeric")
  }

  z<-c()
  i<-1
  while(i<=length(x)-n){
    z[i]<-median(x[seq(from =i,to=i+n)],...) # Here we include it to later be able to call the na.rm=T
    i<-i+1
  }
  return(z)
}


my_moving_median(x = 1:10, n = 2)
```

```
## [1] 2 3 4 5 6 7 8 9
```

```
my_moving_median(x = 5:15, n = 4)
```

```
## [1]  7  8  9 10 11 12 13
```

```
my_moving_median(x = c(5, 1, 2, NA, 2, 5, 6, 8, 9, 9), n = 2)
```

```
## [1]  2 NA NA NA  5  6  8  9
```

```
my_moving_median(x = c(5, 1, 2, NA, 2, 5, 6, 8, 9, 9), n = 2, na.rm = TRUE)
```

```
## [1] 2.0 1.5 2.0 3.5 5.0 6.0 8.0 9.0
```

Question 4

Multiplication table

```
for_mult_table<- function(from,to){

   if(!is.numeric(from)|!is.numeric(to)){
    stop("x or n is not numeric")
  }

Mat<- matrix(nrow =length(from:to),ncol = length(from:to) )

rows<-from:to
cols<-from:to
rownames(Mat)<-rows
colnames(Mat)<-cols

for(i in 1:dim(Mat)[1]){
  for(j in 1:dim(Mat)[2]){
    Mat[i,j]<- rows[i]*cols[j]
  }

}
  return(Mat)

}

for_mult_table(from = 1, to = 5)
```

```
##   1  2  3  4  5
## 1 1  2  3  4  5
## 2 2  4  6  8 10
## 3 3  6  9 12 15
## 4 4  8 12 16 20
## 5 5 10 15 20 25
```

```
for_mult_table(from = 10, to = 12)
```

```
##     10  11  12
## 10 100 110 120
## 11 110 121 132
## 12 120 132 144
```

Question 5

Correlation matrix.

```
cor_matrix<-function(x){

  if(!is.data.frame(x)){
    stop("Data is not a dataframe!!")
  }

  Mat<- scale(x,center = TRUE, scale = TRUE)
  n<-nrow(x)
  correlation_matrix<-(t(Mat)%*%Mat)/(n-1)

  return(correlation_matrix)

}

data(iris)
cor_matrix(iris[, 1:4])
```

```
##              Sepal.Length Sepal.Width Petal.Length Petal.Width
## Sepal.Length    1.0000000  -0.1175698    0.8717538   0.8179411
## Sepal.Width    -0.1175698   1.0000000   -0.4284401  -0.3661259
## Petal.Length    0.8717538  -0.4284401    1.0000000   0.9628654
## Petal.Width     0.8179411  -0.3661259    0.9628654   1.0000000
```

```
cor_matrix(as.list(iris[,1:4]))
```

```
## Error in cor_matrix(as.list(iris[, 1:4])): Data is not a dataframe!!
```

Question 6

Calculating a cumulative sum and stopping when find_sum is traversed.

```
find_cumsum<- function(x,find_sum){

  if(!is.numeric(x)|!is.numeric(find_sum)){
    stop("Arguments are nor numeric!")
  }


  sum1 <- 0
  i<- 1

  while(i<=length(x)){

    sum1<-sum1+i
    i<-i+1
    if(sum1>find_sum)
      break;

  }

  return(sum1)
}

find_cumsum(x = 1:100, find_sum = 500)
```

```
## [1] 528
```

```
find_cumsum(x = 1:10, find_sum = 500)
```

```
## [1] 55
```

Question 7

Multiplication table as in question 4 but with a while loop. Tricky one where you have to think about the columns and rows.

```
while_mult_table<- function(from,to){

  if(!is.numeric(from)|!is.numeric(to)){
    stop("From and to is not numeric")
  }

  rows <- from:to
  cols<- from:to
  mat<- matrix(nrow = length(rows),ncol = length(cols)) # Empty
  rownames(mat)<-rows
  colnames(mat)<-cols

  i<--length(rows)
  j<--length(cols)
  while(i<= dim(mat)[1]){
    while(j<= dim(mat)[2]){

    mat[i,j]<-rows[i]*cols[j]
    j<-j+1
   i<-i
    }
   i<-i+1
   j<-i

  }
  return(mat)

}


while_mult_table(from = 3, to = 5)
```

```
##    3  4  5
## 3  9 12 15
## 4 12 16 20
## 5 15 20 25
```

```
while_mult_table(from = 7, to = 12)
```

```
##     7  8   9  10  11  12
## 7  49 56  63  70  77  84
## 8  56 64  72  80  88  96
## 9  63 72  81  90  99 108
## 10 70 80  90 100 110 120
## 11 77 88  99 110 121 132
## 12 84 96 108 120 132 144
```

Question 8

Same as question 6 but with a repeat loop.

```
repeat_find_cumsum<-function(x,find_sum){
  if(!is.numeric(x)|!is.numeric(find_sum)){
    stop("Arguments are not numeric")
  }

  sum1<- 0
  i<-1

  repeat{

    sum1<-sum1+i
    i<-i+1

    if(i>length(x))
      break()

    if (sum1>find_sum)
      break()

  }
  return(sum1)

}

repeat_find_cumsum(x = 1:100, find_sum = 500)
```

```
## [1] 528
```

```
repeat_find_cumsum(x = 1:10, find_sum = 500)
```

```
## [1] 55
```

Question 9

Moving median again but with a repeat loop.

```
repeat_my_moving_median<- function(x,n,...){
  if(!is.numeric(x)|!is.numeric(n)){
    stop("Arguments are not numeric!")
  }

  z<-c()
  i<-1

  repeat{


    z[i]<- median(x[seq(from=i,to=i+n)],na.rm = TRUE)
    i<-i+1

    if (i>length(x))
      break()
  }

  return(z)
}


repeat_my_moving_median(x=1:10,n=2)
```

```
##  [1]  2.0  3.0  4.0  5.0  6.0  7.0  8.0  9.0  9.5 10.0
```

```
repeat_my_moving_median(x = 5:15, n = 4)
```

```
##  [1]  7.0  8.0  9.0 10.0 11.0 12.0 13.0 13.5 14.0 14.5 15.0
```

Question 10

lapply applies functions on list objects and returns a list object of the same length. A data frame is actually a type of list, so we can use lapply on a data frame without first converting it with as.list()

```
coefvar <- function(X){

  if(!is.data.frame(X)){
    stop("X is not a dataframe")
  }
  z<- c(lapply(X,
               FUN = function(X)
                 sd(X)/mean(X)
               ))

  return(z)
}


data("iris")

coefvar(X=iris[1:4])
```

```
## $Sepal.Length
## [1] 0.1417113
##
## $Sepal.Width
## [1] 0.1425642
##
## $Petal.Length
## [1] 0.4697441
##
## $Petal.Width
## [1] 0.6355511
```

Question 11

Function for calculating bmi, the function gives a warning message if weight or height is <= 0.

```
bmi<-function(body_weight,body_height){

  bmi<- body_weight/body_height^2

  if (body_weight<=0)
    warning("body_weight is not positive, calculation is not meaningful")

   if (body_height<=0)
    warning("body_heigth is not positive, calculation is not meaningful")
}

bmi(body_weight = 95, body_height = 1.98)

bmi(body_weight = 74, body_height = -1.83)
```

```
## Warning in bmi(body_weight = 74, body_height = -1.83): body_heigth is not
## positive, calculation is not meaningful
```

Question 12

Babylon method for approximating the square root of a number.

```
babylon <-  function(x, init, tol) {
  sqrt_approx <- x
  prop <- init
  new <- (prop + x / prop) / 2
  iter <- 0
  while (abs(new - prop) > tol) {
  iter <- iter + 1
  prop <-  new
  new <-  (prop + x / prop) / 2
  rot <-  new
  print(prop)
  }
  return(list(Iterations = iter, Sqrt = rot))
}


babylon(40, 20, 0.1)
```

```
## [1] 11
## [1] 7.318182
## [1] 6.39201
```

```
## $Iterations
## [1] 3
##
## $Sqrt
## [1] 6.324911
```

```
babylon(x = 2, init = 1.5, tol = 0.01)
```

```
## [1] 1.416667
```

```
## $Iterations
## [1] 1
##
## $Sqrt
## [1] 1.414216
```

Question 13

Hilbert matrix with nested for loop.

```
hilbert_matrix <- function(nrrow,nrcol){

  Mat<-matrix(nrow = nrrow,ncol = nrcol)

  for(i in 1:nrrow){
   for(j in 1:nrcol)
     Mat[i,j]<- 1/(i+j-1)
  }
 return(Mat)
}

hilbert_matrix(nrrow=5, nrcol=2)
```

```
##           [,1]      [,2]
## [1,] 1.0000000 0.5000000
## [2,] 0.5000000 0.3333333
## [3,] 0.3333333 0.2500000
## [4,] 0.2500000 0.2000000
## [5,] 0.2000000 0.1666667
```

```
hilbert_matrix(nrrow=5, nrcol=4)
```

```
##           [,1]      [,2]      [,3]      [,4]
## [1,] 1.0000000 0.5000000 0.3333333 0.2500000
## [2,] 0.5000000 0.3333333 0.2500000 0.2000000
## [3,] 0.3333333 0.2500000 0.2000000 0.1666667
## [4,] 0.2500000 0.2000000 0.1666667 0.1428571
## [5,] 0.2000000 0.1666667 0.1428571 0.1250000
```

Question 14

Toeplitz matrix

```
toeplitz_matrix <- function(x) {
  if (length(x) %% 2 == 0) {
    stop("Length of vector x is even")
  }
  res_mat <-matrix(nrow = ceiling(length(x) / 2), ncol = ceiling(length(x) / 2))
  n <-  ceiling(length(x) / 2)
  res_mat[1, ] <- x[1:n]
  res_mat[-1, 1] <- x[(n + 1):length(x)]
  for (i in 2:n) {
    for (j in 2:n) {
      res_mat[i, j] <- res_mat[i - 1, j - 1]


    }


  }
  return(res_mat)
}



toeplitz_matrix(x = 1:5)
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    1    2
## [3,]    5    4    1
```

```
toeplitz_matrix(x = 1:4)
```

```
## Error in toeplitz_matrix(x = 1:4): Length of vector x is even
```

```
toeplitz_matrix(c(1, 0, 2, 0, 3, 0, 4, 0, 5))
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    0    2    0    3
## [2,]    0    1    0    2    0
## [3,]    4    0    1    0    2
## [4,]    0    4    0    1    0
## [5,]    5    0    4    0    1
```